TLS
Internet-Draft
Intended status: Standards Track
Expires: 23 April 2026

N. Sullivan
Cryptography Consulting LLC
D. Jackson
Mozilla

20 October 2025

# Authenticated ECH Config Distribution and Rotation
draft-sullivan-tls-signed-ech-updates-00

## Abstract

Encrypted ClientHello (ECH) requires clients to have the server’s ECH configuration before connecting. Currently, when ECH fails, servers can send updated configurations but clients cannot authenticate them without a certificate for the public name, limiting deployment flexibility.

This document specifies an authenticated ECH configuration update mechanism. Servers can deliver signed ECH configurations during the TLS handshake, allowing clients to authenticate and immediately use them for retry. The mechanism decouples ECH key distribution from transport, enabling the same signed configuration to work via DNS or TLS delivery.

## About This Document

This note is to be removed before publishing as an RFC.

Status information for this document may be found at https://datatracker.ietf.org/doc/draft-sullivan-tls-signed-ech-updates/.

Source for this draft and an issue tracker can be found at https://github.com/grittygrease/draft-sullivan-tls-signed-ech-updates.

## Status of This Memo

## Copyright Notice



## Table of Contents

## 1. Introduction

Deployment of TLS Encrypted ClientHello (ECH) requires that clients obtain the server's current ECH configuration (ECHConfig) before initiating a connection. Current mechanisms distribute ECHConfig data via DNS HTTPS resource records [RFC9460] or HTTPS well-known URIs [I-D.ietf-tls-wkech], allowing servers to publish their ECHConfigList prior to connection establishment.

While ECH includes a retry mechanism where servers can send updated ECHConfigList values during the handshake (via HelloRetryRequest or EncryptedExtensions), the base ECH specification instructs clients not to cache these configurations [I-D.ietf-tls-esni]. Instead, servers must authenticate the outer handshake using a certificate for the public name, and clients must obtain updated configurations through out-of-band mechanisms for future connections.

This approach limits ECH deployment in two key ways. First, it restricts the usable public names to those for which operators can obtain certificates, reducing the potential anonymity set. Second, it creates delays in key rotation recovery, as clients cannot immediately use updated configurations received during the handshake.

This document introduces an Authenticated ECH Config Update mechanism to securely deliver and rotate ECHConfig data in-band, during a TLS handshake. The goal is to allow servers to frequently update ECH keys (for example, to limit the lifetime of keys or respond to compromise). This mechanism does not prescribe or rely on fallback to cleartext; when ECH fails with this mechanism, the connection is terminated and retried with updated configurations rather than exposing the protected name.

The mechanism supports two authentication methods through the ech_auth ECHConfig extension:

1. Raw Public Key (RPK) - SPKI hashing for configuration-specific trust

2. PKIX - Certificate-based signing with a critical X.509 extension

Each ECHConfig carries at most one signature using the specified method. By authenticating ECH configs independently, the mechanism makes ECH key distribution orthogonal to transport. The same signed ECHConfig artifact can be conveyed via DNS, HTTPS, or the TLS handshake itself. The client will accept it only if the accompanying signature or proof is valid under one of its trust modes.

## 2. Conventions and Definitions

The key words "MUST", "MUST NOT", "SHOULD", "SHOULD NOT", and "MAY" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals.

This document assumes familiarity with TLS 1.3 [RFC8446] and the ECH specification [I-D.ietf-tls-esni], referred to here as simply "ECH".

The following acronyms are used throughout this document:

* RPK: Raw Public Key - A public key used directly without a certificate wrapper

* PKIX: Public Key Infrastructure using X.509 - The standard certificate-based PKI used on the web

* SPKI: SubjectPublicKeyInfo - The ASN.1 structure containing a public key and its algorithm identifier

* HPKE: Hybrid Public Key Encryption - The encryption scheme used by ECH as defined in [RFC9180]

* DER: Distinguished Encoding Rules - A binary encoding format for ASN.1 structures

* OCSP: Online Certificate Status Protocol - A method for checking certificate revocation status

* CRL: Certificate Revocation List - A list of revoked certificates published by a Certificate Authority

* CA: Certificate Authority - An entity that issues digital certificates

## 2.1. Terminology

ECHConfig: An individual ECH configuration structure as defined in [I-D.ietf-tls-esni], which includes fields such as public_name, public_key (HPKE key), and extensions.

ECHConfigList: A sequence of one or more ECHConfig structures as defined in ECH (a byte string that starts with a 16-bit length and may contain multiple concatenated ECHConfig values).

ECHConfigTBS (To-Be-Signed): The serialized ECHConfig structure with the ech_auth extension, but with the signature field within ech_auth set to zero-length. This includes all ECHConfig fields and the ech_auth extension's method and trusted_keys fields.

authenticated ECHConfig: An ECHConfig that contains an ech_auth extension with a valid signature in the signature field, allowing clients to verify its authenticity.

public name: The value of the public_name field in the ECHConfig, i.e., the authoritative DNS name for updates and validation associated with that configuration. This name is not required to be the ClientHelloOuter SNI, though deployments sometimes choose to align them.

retry_configs: The ECHConfigList sent by a server in HelloRetryRequest or EncryptedExtensions when ECH is rejected, as defined in [I-D.ietf-tls-esni].

outer SNI: The Server Name Indication value sent in the outer (unencrypted) ClientHello when ECH is used. This is typically the ECHConfig's public_name or another name that preserves client privacy.

The reader should recall that in TLS 1.3, the server's EncryptedExtensions message is encrypted and integrity-protected with handshake keys [I-D.ietf-tls-esni]. New extensions defined as part of EncryptedExtensions are not visible to network attackers and cannot be modified by an attacker without detection. Additionally, "certificate verification" refers to the standard X.509 validation process (chain building, signature and expiration checking, name matching, etc.) unless otherwise specified.

## 3. Mechanism Overview

This specification defines three methods for authenticating ECH configuration updates:

### 3.1. Raw Public Key (RPK)

The ECHConfigList update is authenticated by a Raw Public Key (RPK). The ECHConfig's ech_auth extension carries a set of trusted_keys, each value being SHA-256(SPKI) of an RPK that is authorized to sign an update.

A client receiving an authenticated update (e.g., in HRR/EE retry_configs) MUST:

1. Extract the authenticator key's SubjectPublicKeyInfo (SPKI) and compute sha256(spki). Verify membership in trusted_keys.

2. Verify that not_after is strictly greater than the client's current time.

3. Verify the signature over the to_be_signed input using the authenticator key.

Clients MAY cache trusted_keys only for the lifetime of the associated ECHConfig and solely to authenticate a single retry-configs update per fresh connection attempt. Upon successful installation of a new ECHConfigList (via an authenticated update or a fresher out-of-band fetch), or upon expiration of the cached configuration, clients MUST clear any RPK state learned from the prior configuration.

This specification defines no long-term pinning and no public-name authentication via RPK.

### 3.2. PKIX (Certificate-Based)

The update is signed with the private key corresponding to an X.509 certificate that chains to a locally trusted root and is valid for the ECHConfig public_name (i.e., appears in the certificate's SAN).

The leaf certificate MUST include a new, critical X.509 v3 extension id-pe-echConfigSigning (OID: TBD) whose presence indicates authorization to sign ECH configuration updates for the DNS names in the certificate's SAN. Clients:

* MUST validate the certificate chain according to local policy;

* MUST confirm the SAN covers the ECHConfig public_name;

* MUST confirm the critical id-pe-echConfigSigning extension is present in the leaf; and

* MUST verify the ECH update signature with the leaf key.

When this critical extension is present, clients MUST NOT accept the certificate for TLS server authentication. The not_after field in ech_auth.signature MUST be 0 for PKIX.

## 4. Benefits of Signed ECH Configurations

By treating ECH configurations as signed objects, this mechanism decouples trust in ECH keys from the TLS handshake's certificate validation of the origin. This enables several important capabilities:

### 4.1. Distinct Public Names Without CA Certificates

A server can use many different public hostnames (even per-client, per-connection unique ones) to maximize the anonymity set or for other operational reasons [I-D.ietf-tls-esni], without having to obtain certificates for each. The RPK method allows the client to authenticate the server's ability to update ECH keys for those public names via a key hash, rather than via a CA-issued certificate. This was not possible under the original ECH design, which required a valid certificate for any public name used [I-D.ietf-tls-esni].

### 4.2. Faster and Safer Key Rotation

The server can proactively push a new ECHConfig to clients shortly before rotating keys, ensuring clients receive it immediately. The update objects include an expiration timestamp (not_after), allowing servers to bound the lifetime of configurations to coordinate key rollover. Clients will reject expired configurations.

### 4.3. Out-of-Band Distribution Synergy

Because the same authentication methods are defined for in-band and out-of-band, an ECHConfig obtained via DNS can carry the same signature that a TLS in-band update would, allowing clients to verify it even if their DNS channel is not fully trusted. For instance, a client might obtain an ECHConfig via DNS; if that ECHConfig has RPK trusted_keys, subsequent updates via TLS will be signed by that key, protecting against any earlier undetected DNS tampering.

### 4.4. Design Simplicity

This design attempts to minimize complexity. It does not use explicit key identifiers or complicated pin rotation metadata. For the RPK method, the pinning model is kept simple (a list of allowed signing keys established from an authenticated initial configuration); pin revocation or addition is handled by simply signing a new update that changes the list (clients trust the new list if it is signed by a currently trusted key). There is no "next update time" field that requires clients to preemptively fetch updates; instead, updates are fetched when provided by the server or when the client next connects. The mechanism is agnostic to the transport by which the client obtained the initial ECHConfig whether via DNS SVCB/HTTPS RR (as in [I-D.ietf-tls-svcb-ech]), via a well-known HTTPS endpoint [I-D.ietf-tls-wkech], or via some provisioning protocol, the subsequent updates use the same verification process.

## 5. Protocol Elements

This section specifies the new extensions and data structures in detail. All multi-byte values are in network byte order (big-endian). The syntax uses the TLS presentation language from [RFC8446].

### 5.1. ECH authentication extension (ech_auth)

The ech_auth information is carried as an ECHConfig extension inside the ECHConfig structure and is used both when distributed via DNS and when delivered in TLS (HRR/EE) as part of retry_configs. This single extension conveys policy (which signature methods and pins are supported) and, when present, a signed authenticator that allows clients to verify and install the ECHConfig immediately.

The ech_auth extension MUST be the last extension in the ECHConfig's extension list. This ensures that the signature in the extension covers all other extensions in the ECHConfigTBS. Implementations MUST place this extension last when constructing an ECHConfig, and MUST reject ECHConfigs where ech_auth is not the last extension.

The ech_auth extension has the following structure:

```
enum {
    none(0),
    rpk(1),
    pkix(2),
    (255)
} ECHAuthMethod;

// We reuse the TLS HashAlgorithm registry values (though TLS 1.3 itself
// doesn't use this enum directly, the registry still exists)
// For now, implementations MUST use sha256(4). Future specs may allow others.
opaque SPKIHash<32..32>;  // SHA-256 hash of DER-encoded SPKI

   struct { ECHAuthMethod method; // Single authentication method
   SPKIHash trusted_keys<0..2^16-1>; // RPK-only; SHA-256 hashes per
   IANA TLS // HashAlgorithm registry value 4; // zero-length if method
   != rpk

  // Optional signed authenticator. Present when the sender wishes
  // to provide a signed ECHConfig (e.g., in TLS retry_configs, or
  // pre-signed in DNS). struct {
    opaque authenticator<1..2^16-1>; // method-specific material (see below)
    uint64 not_after;                // Unix timestamp; used by RPK;
                                     // MUST be 0 for PKIX SignatureScheme algorithm;
    opaque signature<1..2^16-1>;
  } signature;                       // Optional; zero-length if not present
} ECHAuth;
```

#### 5.1.1. Signature Computation

The signature is computed over the concatenation:

```
context_label = "TLS-ECH-AUTH-v1"  // ASCII, no NUL
to_be_signed = context_label || ECHConfigTBS
```

where:

* ECHConfigTBS (To-Be-Signed) is the serialized ECHConfig structure including the ech_auth extension, but with the signature field within ech_auth set to zero-length. This means it includes:
  - All ECHConfig base fields (version, length, contents, etc.)
  - All extensions including ech_auth (which MUST be last)
  - Within ech_auth: the method, trusted_keys, and the authenticator/ not_after/algorithm fields from signature, but NOT the actual signature bytes
* The signature is computed over this entire structure, avoiding circular dependency by zeroing out only the signature bytes themselves
* All multi-byte values use network byte order (big-endian)
* The serialization follows TLS 1.3 presentation language rules from RFC 8446

Including a fixed, scheme-specific context label prevents cross-protocol reuse; covering the to-be-signed ECHConfig and all ech_auth fields (except the signature itself) ensures integrity of parameters and pins. The not_after timestamp provides freshness by bounding the configuration’s validity period.

Method-specific authenticator:

* RPK (method=1): the DER-encoded SubjectPublicKeyInfo (SPKI) of the signing key. The client MUST compute the SHA-256 hash of the SPKI, verify that it matches one of the hashes in trusted_keys, check that the current time is before the not_after timestamp, and then verify the signature with this key. The not_after field is REQUIRED and MUST be a timestamp strictly greater than the client’s current time at verification.

* PKIX (method=2): a CertificateEntry vector (leaf + optional intermediates) as in TLS 1.3 Certificate; the leaf MUST include the critical id-pe-echConfigSigning extension and be valid for the ECHConfig public_name. The client validates the chain, confirms the SAN includes the ECH public_name, confirms the critical id-pe-echConfigSigning extension is present in the leaf, and verifies the signature with the leaf key. The not_after field MUST be set to 0 (absent).

Notes:

* trusted_keys is only used by RPK; clients MUST ignore it for PKIX.

* If method is rpk(1), trusted_keys MUST contain at least one SPKI hash; otherwise it MUST be zero-length.

* A server publishing multiple ECHConfigs MAY use different methods for each to maximize client compatibility.

Context-specific requirements:

* When carried in TLS (HelloRetryRequest or EncryptedExtensions), an ech_auth extension in each delivered ECHConfig MUST include a signed authenticator in signature, and the client MUST verify the authenticator before installing the ECHConfig.

* When carried in DNS, an ech_auth extension MAY omit the signature field (unsigned), in which case it conveys only policy (method, trusted_keys). Clients MAY use such information to attempt ECH and to bootstrap trust, but MUST NOT treat it as an authenticated update. If signature is present in DNS, clients SHOULD verify it per the indicated method and MAY treat the ECHConfig as authenticated upon successful verification.

The SPKI hash uses SHA-256 (value 4 in the IANA TLS HashAlgorithm registry). The rationale for using a hash rather than the full SPKI is to keep the extension compact in DNS and on the wire, and to avoid exposing full public keys in the clear. SHA-256 is universally supported and provides sufficient security. The drawback is that hash collisions or second-preimage attacks could undermine the pin this is considered cryptographically infeasible for SHA-256 at the time of writing.

Note: While TLS 1.3 moved to SignatureScheme and doesn’t directly use the HashAlgorithm enum, we reference the IANA registry value for clarity. Future versions of this specification could add a hash algorithm field using the TLS HashAlgorithm registry if algorithm agility becomes necessary.

Client behavior: When a client obtains an ECHConfig that contains an ech_auth extension, it SHOULD store this information along with the configuration. If the client subsequently uses this ECHConfig to initiate a connection, it relies on delivery of signed ECHConfigs in HRR/EE for in-band updates.

If an ECHConfig does not include ech_auth, the in-band update mechanism defined here is not used for that configuration.

Server behavior: A server that wishes to allow in-band updates MUST include ech_auth in the ECHConfig it publishes via DNS or other means. The server MUST set the method field to the authentication method it will use for this configuration. The server MUST ensure that it actually has the capability to perform the indicated method:

* If method is rpk(1), the server needs a signing key whose SPKI hash is in trusted_keys. (It may have multiple keys for rotation; all keys that might sign an update before the next ECHConfig change should be listed. Pins can be added or removed by generating a new ECHConfig with an updated list and distributing it out-of-band or via an update.)

* If method is pkix(2), the server must have a valid certificate (and chain) for the public name with the critical id-pe-echConfigSigning extension (Section IANA Considerations (Section 9) defines the extension) available at runtime to use for signing. The certificate’s public key algorithm dictates what signature algorithms are possible.

## 5.2. TLS Extensions for ECH Config Update

### 5.2.1. EncryptedExtensions / HelloRetryRequest Delivery

This specification reuses the ECH retry_configs delivery mechanism: the server sends an ECHConfigList where each ECHConfig contains the ech_auth extension with a signed authenticator. The server MAY include multiple ECHConfigs with different authentication methods (e.g., one with PKIX and one with RPK) to maximize client compatibility. There is no separate TLS extension for negotiation.

### 5.2.2. Server Behavior

When a server receives a ClientHello with the encrypted_client_hello extension, it processes ECH per [I-D.ietf-tls-esni]. If the server has an updated ECHConfigList to distribute:

1. ECH Accepted: If the server successfully decrypts the ClientHelloInner, it completes the handshake using the inner ClientHello. The server MAY include authenticated ECHConfigs in EncryptedExtensions if an update is available.

2. ECH Rejected: If the server cannot decrypt the ClientHelloInner, it SHOULD proceed with the outer handshake and include authenticated ECHConfigs in EncryptedExtensions. This allows the client to immediately retry with the correct configuration.

The server prepares authenticated updates by:

* Using the authentication method specified in the ECHConfig's ech_auth.method

* Creating the appropriate authenticator (RPK signature or PKIX certificate chain)

* Including the authenticator in the ECHConfig's ech_auth.signature field

* Sending the ECHConfigList via the existing retry_configs mechanism

#### 5.2.3. Client Behavior

When a client retrieves an ECHConfig (e.g., from DNS), it examines the ech_auth extension and records:

* The authentication method (RPK or PKIX)

* Any trusted_keys for RPK validation

* Any pre-distributed signature for immediate validation

During the TLS handshake, upon receiving an ECHConfigList in HRR or EE:

1. Validation: The client validates the authenticator according to its method:

    * RPK: Computes the SHA-256 hash of the provided SPKI, verifies it matches one in trusted_keys, then verifies the signature

    * PKIX: Validates the certificate chain, verifies the leaf certificate covers the ECHConfig's public_name, checks for the critical id-pe-echConfigSigning extension, then verifies the signature

2. Validity Checking: The client checks temporal validity:

    * For RPK: Verifies current time is before not_after

    * For PKIX: Verifies certificate validity period (the not_after field MUST be 0)

3. Installation and Retry (see Appendix A for state diagram):

    * If validation succeeds and this was an ECH rejection (outer handshake):

        - The client treats the retry_configs as authentic per [I-D.ietf-tls-esni], Section 6.1.6

        - The client MUST terminate the connection and retry with the new ECHConfig

        - The retry does not consider the server’s TLS certificate for the public name

    * If validation succeeds and this was an ECH acceptance:

        - The client caches the new ECHConfig for future connections

    * If validation fails:

        - The client MUST treat this as if the server’s TLS certificate could not be validated

        - The client MUST NOT use the retry_configs

        - The client terminates the connection without retry

Note: Regardless of validation outcome in an ECH rejection, the client will terminate the current connection. The difference is whether it retries with the new configs (validation success) or treats it as a certificate validation failure (validation failure). Implementers should refer to the state diagram in Appendix A for the complete retry logic flow.

#### 5.2.4. Backward Compatibility

Clients that do not implement this specification continue to process retry_configs as defined in [I-D.ietf-tls-esni], ignoring the authentication extensions. Servers that do not implement this specification send unauthenticated retry_configs as usual.

# 6. Example Exchange

## 6.1. Initial Setup

Consider api.example.com as a service protected by ECH with public name ech.example.net. The operator publishes an ECHConfig via DNS HTTPS RR with the ech_auth extension containing:

* Method: RPK (value 1)
* Trusted keys: SHA-256 hash of an Ed25519 signing key's SPKI
* Optional: A signed authenticator for immediate validation

## 6.2. Successful ECH with Update

1. Client connects: Sends ClientHello with ECH using cached config
    * Outer SNI: ech.example.net
    * Inner SNI: api.example.com
2. Server accepts ECH: Decrypts inner ClientHello successfully
    * Prepares updated ECHConfig with new keys
    * Selects PKIX method for signing
    * Signs the update with certificate containing the critical id-pe-echConfigSigning extension
3. Server response:
    * ServerHello (ECH accepted)
    * EncryptedExtensions containing authenticated ECHConfig
    * Certificate for api.example.com (inner origin)
    * CertificateVerify, Finished
4. Client validation:
    * Verifies ECH acceptance
    * Validates PKIX certificate chain and critical extension
    * Verifies signature over ECHConfig

* Caches new config for future connections

### 6.3. ECH Rejection with Recovery

1. Client connects: Uses outdated ECHConfig
2. Server rejects ECH: Cannot decrypt inner ClientHello
3. Server continues outer handshake:
    * Sends authenticated ECHConfig in EncryptedExtensions
    * Uses certificate for ech.example.net
4. Client recovery:
    * Validates and caches new ECHConfig
    * Closes connection (not authenticated for inner origin)
    * Immediately retries with new ECHConfig

## 7. Security Considerations

This section analyzes security properties by threat model.

### 7.1. Passive Attackers

This mechanism preserves ECH's protection against passive observation. ECHConfig updates are delivered within encrypted TLS messages (HelloRetryRequest or EncryptedExtensions), preventing passive observers from learning about configuration changes. The mechanism ensures that even during retry scenarios, the client's intended server name is never exposed in cleartext.

### 7.2. Active Network Attackers

#### 7.2.1. Initial Trust Bootstrap

The security of this mechanism fundamentally depends on the authenticity of the initial ECHConfig. If an attacker can inject a malicious initial configuration, they may be able to pin their own keys in the trusted_keys field, enabling persistent interception.

When ECHConfigs are obtained via DNS, an on-path attacker could provide a fake ECHConfig with the attacker's key in trusted_keys. While the attacker cannot decrypt ECH-protected connections, they could cause ECH to fail and then present their own certificate during

fallback, potentially tricking the client into accepting and caching a compromised configuration. Clients SHOULD prefer authenticated bootstrap mechanisms when available.

Initial retrieval of ECHConfigList via DNS is unchanged by this mechanism. This specification authenticates updates in-band via RPK or PKIX; it does not attempt to authenticate the initial DNS fetch. ECHConfigs obtained via HTTPS from a well-known URI benefit from Web PKI authentication. Pre-configured ECHConfigs in applications derive their trust from the application's distribution channel.

#### 7.2.2. Handshake Integrity

Signed ECHConfigs delivered via HelloRetryRequest or EncryptedExtensions are protected by TLS 1.3's handshake encryption and integrity mechanisms. The Finished message ensures that any modification by an attacker would be detected.

A man-in-the-middle attacker without the server's handshake keys cannot modify the EncryptedExtensions message containing the ECHConfig update. The TLS 1.3 handshake itself provides replay protection through its use of fresh random values and the Finished message authentication.

### 7.3. Compromised Keys

#### 7.3.1. Signature Verification

Clients MUST correctly implement signature verification for each authentication method. For RPK, servers and clients SHOULD use cryptographically strong signature schemes from the TLS 1.3 SignatureScheme registry, such as Ed25519, ECDSA, or RSA-PSS. Weak schemes like RSA PKCS#1 v1.5 SHOULD NOT be used.

The inclusion of not_after timestamps (for RPK) or certificate validity periods (for PKIX) ensures configuration freshness. These temporal bounds prevent clients from accepting stale configurations that might use compromised keys or outdated parameters. Clients MUST verify these temporal constraints and reject expired configurations. Note that these mechanisms depend on reasonably synchronized clocks (within 5 minutes of actual time is RECOMMENDED).

Note that signed ECHConfigs themselves are replayable - an attacker could capture and resend a valid signed configuration. However, this is not a security concern as the configuration is public data intended for distribution. The freshness guarantees ensure that old configurations eventually expire, limiting the window during which outdated keys remain acceptable.

#### 7.3.2. Key Management

Servers MUST protect their ECH update signing keys. If an RPK signing key is compromised, the server SHOULD remove its hash from trusted_keys in subsequent updates, signing the transition with a different trusted key. Servers SHOULD consider including multiple keys in trusted_keys to facilitate key rotation and recovery from compromise.

For PKIX-based updates, normal certificate lifecycle management applies. Servers SHOULD obtain new certificates before existing ones expire.

For PKIX authentication, this specification leverages existing CA infrastructure including revocation mechanisms. A compromised ECH signing certificate could be used to sign malicious updates, but this risk is mitigated by the certificate's constraints (critical extension and name binding) and standard revocation mechanisms (OCSP/CRL). Clients SHOULD apply the same revocation checks to ECH signing certificates as they do for TLS server certificates.

### 7.4. Implementation Vulnerabilities

#### 7.4.1. Failure Handling

When ECHConfig update verification fails, clients MUST NOT compromise the security or privacy guarantees of ECH. If ECH was accepted but the update verification failed, the connection proceeds normally without caching the new configuration. This represents a safe failure mode where connectivity is maintained but key rotation is delayed.

If ECH was rejected and the update verification also fails, the client lacks a valid configuration for retry. In this case, the client SHOULD NOT proceed with the connection using the outer SNI for application data, as this would violate ECH's privacy goals. The client MAY attempt to obtain a valid configuration through other means (such as DNS) or treat the connection as failed.

Servers MAY include multiple ECHConfigs with different authentication methods to maximize the probability of successful verification. Clients SHOULD process these in order and use the first configuration that successfully verifies.

In distributed deployments, only servers with access to the appropriate signing keys can generate valid ECHConfig updates. This prevents unauthorized intermediaries (such as CDN nodes) from injecting malicious configurations. If a server sends an update that

cannot be verified, the client simply ignores it and continues with its existing configuration. While this could potentially lead to the use of outdated configurations, it prevents compromise of the ECH mechanism itself.

Algorithm agility is provided through the TLS SignatureScheme registry for RPK and standard PKIX certificate algorithms. Implementations SHOULD support commonly deployed algorithms and MUST be able to handle algorithm transitions.

#### 7.4.2. Denial of Service Considerations

Signature verification introduces computational costs. However, these operations occur only during ECH configuration updates, not on every connection. The additional data in EncryptedExtensions (certificates) may increase message sizes, potentially causing fragmentation in some scenarios. Implementations SHOULD be aware of message size limits, particularly in QUIC deployments.

Attackers cannot force servers to send signed ECHConfigs without establishing TLS connections. Standard TLS denial-of-service mitigations (rate limiting, stateless cookies) apply equally to this mechanism.

## 8. Privacy Considerations

This mechanism preserves and potentially enhances ECH’s privacy properties. By enabling the use of diverse public names through RPK authentication, servers can increase the anonymity set beyond what is possible with certificate-based authentication alone.

The ECHConfig updates themselves are delivered within encrypted TLS messages (HelloRetryRequest or EncryptedExtensions), preventing passive observers from learning about configuration changes. The mechanism ensures that even during retry scenarios, the client’s intended server name is never exposed in cleartext.

A potential privacy consideration is that failed ECH attempts followed by successful retries create a distinctive connection pattern. However, this pattern only reveals that ECH was used with a particular public name, not the intended destination behind that name.

This specification introduces no new tracking mechanisms or identifiers beyond those already present in TLS and DNS.

## 9. IANA Considerations

### 9.1. ECHConfig Extension

IANA is requested to add the following entry to the "ECH Configuration Extension Type Values" registry:

* Extension Name: ech_auth
* Value: TBD1
* Purpose: Conveys supported authentication methods, trusted keys, and optional signed authenticators
* Reference: This document

### 9.2. X.509 Certificate Extension OID

IANA is requested to allocate an object identifier (OID) under the "SMI Security for PKIX Certificate Extensions (1.3.6.1.5.5.7.1)" registry with the following values:

* OID: id-pe-echConfigSigning (1.3.6.1.5.5.7.1.TBD2)
* Name: ECH Configuration Signing
* Description: Indicates that the certificate’s subject public key is authorized to sign ECH configuration updates for the DNS names in the certificate’s Subject Alternative Name (SAN).
* Criticality: Certificates containing this extension MUST mark it critical.
* Reference: This document.

### 9.3. ECH Authentication Methods Registry

IANA is requested to establish a new registry called "ECH Authentication Methods" with the following initial values:

| Value | Method | Description | Reference |
|---|---|---|---|
| 0 | Reserved | Not used | This document |
| 1 | RPK | Raw Public Key | This document |
| 2 | PKIX | X.509 with critical id-pe-echConfigSigning | This document |
| 3-255 | Unassigned | - | - |

Table 1

New values are assigned via IETF Review.

## 10. Deployment Considerations

### 10.1. Method Selection

Operators SHOULD support at least one widely implemented method. PKIX (critical extension) provides easier operational deployment with standard certificate issuance workflows and no client pin state. RPK offers small artifacts and simple verification; any client-side state is bounded to the lifetime of the current ECHConfig and a single authenticated retry per connection attempt.

### 10.2. Size Considerations

When sending authenticated ECHConfigs in HelloRetryRequest, servers should be mindful of message size to avoid fragmentation or exceeding anti-amplification limits. RPK signatures are typically more compact than PKIX certificate chains.

### 10.3. Key Rotation

Publish updates well in advance of key retirement. Include appropriate validity periods for each method. Consider overlapping validity windows to allow graceful client migration.

### 10.4. Pin Management

For RPK deployments:

* Maintain multiple valid pins to enable recovery from key compromise

* Remove compromised pins via authenticated updates signed by remaining trusted keys

* Consider using PKIX to re-establish trust if all pinned keys are compromised

### 10.5. Update Consistency

If sending updates in both HRR and EncryptedExtensions, ensure consistency to avoid client confusion. When possible, send updates in only one location per handshake.

## 11. Acknowledgments

The authors thank Martin Thomson for earlier contributions and discussions on the initial draft.

## 12. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

[RFC9180] Barnes, R., Bhargavan, K., Lipp, B., and C. Wood, "Hybrid Public Key Encryption", RFC 9180, DOI 10.17487/RFC9180, February 2022, <https://www.rfc-editor.org/info/rfc9180>.

[RFC9460] Schwartz, B., Bishop, M., and E. Nygren, "Service Binding and Parameter Specification via the DNS (SVCB and HTTPS Resource Records)", RFC 9460, DOI 10.17487/RFC9460, November 2023, <https://www.rfc-editor.org/info/rfc9460>.

[I-D.ietf-tls-esni] Rescorla, E., Oku, K., Sullivan, N., and C. A. Wood, "TLS Encrypted Client Hello", Work in Progress, Internet-Draft, draft-ietf-tls-esni-25, 14 June 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-tls-esni-25>.

[I-D.ietf-tls-svcb-ech]
Schwartz, B. M., Bishop, M., and E. Nygren, "Bootstrapping TLS Encrypted ClientHello with DNS Service Bindings", Work in Progress, Internet-Draft, draft-ietf-tls-svcb-ech-08, 16 June 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-tls-svcb-ech-08>.

[I-D.ietf-tls-wkech]
Farrell, S., Salz, R., and B. M. Schwartz, "A well-known URI for publishing service parameters", Work in Progress, Internet-Draft, draft-ietf-tls-wkech-10, 19 October 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-tls-wkech-10>.

## Authors' Addresses

Nick Sullivan
Cryptography Consulting LLC
Email: nicholas.sullivan+ietf@gmail.com

Dennis Jackson
Mozilla
Email: ietf@dennis-jackson.uk